ESTRUCTURAS DE DATOS

Curso 2018/19

PRÁCTICA 5

Tablas de dispersión

Instrucciones

- *Se debe completar en una sesión.*
- *Práctica individual.*
- *Lee el enunciado completo antes de comenzar. Los comentarios incluidos en el código también pueden proporcionar información útil.*
- *Se habilitará una tarea para que entregues el código desarrollado.*
- *La práctica será APTA si se superan todos los test de validación proporcionados.*

Estamos escribiendo la lase **`EDLinkedHashSet<T>`** que implemente la interfaz **`Set<T>.`** Internamente usará una tabla de dispersión con resolución de colisiones por direccionamiento abierto. Esta implementación La clase tendrá la peculiaridad de que se mantendrá el orden de inserción de los elementos. Para lograr esté fin cada celda de la tabla de dispersión almacenará una referencia a un nodo doblemente enlazado. Estos enlaces llevarán a otros nodos según su orden de inserción.

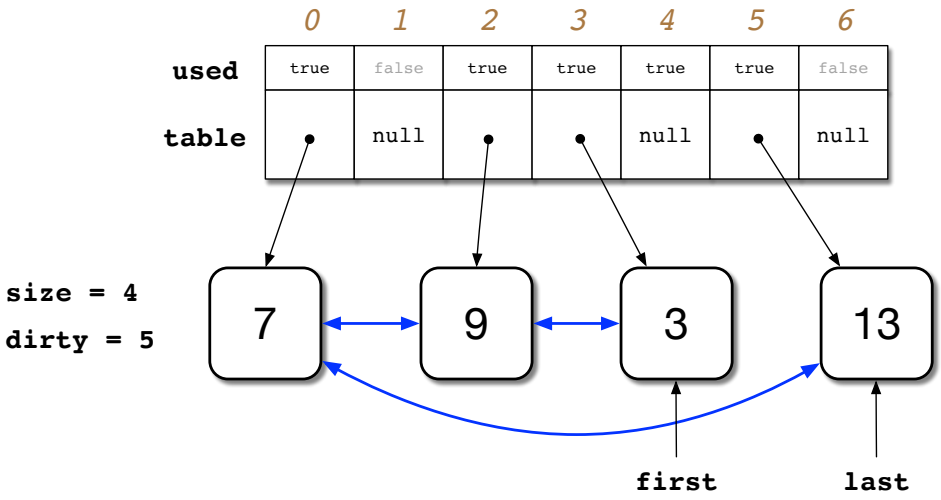
La parte privada de la clase contiene las siguientes definiciones.

```
class Node {
    T data;
    Node next = null;
    Node prev = null;
}

Node[] table;
boolean[] used;
int size = 0;
int dirty = 0;
int rehashThreshold;
Node first = null;
Node last = null;
```

- **`Node`**: clase privada que almacenará los datos del conjunto. Como deberán mantener el orden mantienen un doble enlace a los dato anterior y al posterior.
- **`table`**: la tabla de dispersión. En cada celda almacena una referencia a un nodo.
- **`used`**: tabla de booleanos indicando si la celda correspondiente de table ha sido usada.
- **`size`**: cantidad de datos válidos almacenados en la tabla.
- **`dirty`**: cantidad de casillas de table usadas.
- **`rehashThreshold`**: umbral en el que se realizará el rehash (dirty >= rehashThreshold).
- **`first`**: enlace al nodo con el primer datos insertado en el conjunto.
- **`last`**: Enlace al nodo con el último dato insertado ene l conjunto.

Como ejemplo está sería una imagen de la estructura para en el caso de haber insertado los enteros 3, 9 10, 7, y 13 y haber borrado el 10.



En **`table`** se guardan enlaces a los nodos que almacenan los datos, o **`null`** si no hay nodo almacenado en esas casillas. **`used`** indica las casillas que ha sido usadas. Es el caso de la casilla 4 que es donde se almacenaba la referencia al nodo con el 10 antes de ser borrado. Finalmente, los nodos están encadenados entre sí siguiendo el orden en el que fueron insertados.

El fichero **`EDLinkedHashSet.java`** proporciona la clase parcialmente implementada.

Ejercicio 1

Escribe el método publico **`boolean remove(T item)`** de la clase **`EDLinkedHashList.`** Busca y borra el elemento indicado. Devuelve un valor booleano indicando si ha conseguido borrarlo

Ejercicio 2

Escribe el método privado **`void rehash()`** de la clase **`EDLinkedHashList.`** El método comprueba si **`dirty >= rehashThreshold.`** En caso de que sea así dobla el tamaño de **`table`** y redispersa todos los elementos.

NOTA: La clase **`EDLinkedHashSet`** que se te proporciona tan sólo implementa los métodos **`size`**, **`isEmpty`**, **`clear`**, **`add`**, **`hash`**, **`toArray`**, **`toString`** y un constructor de copia. Si necesitas alguno de los otros métodos, deberás implementarlos.

NOTA 2: Para reservar un vector de **`Node`** es necesario usar la expresión:

```
new EDLinkedHashSet.Node[tamaño];
```

Cualquier otra expresión es errónea. Esta expresión ya aparece así en los constructores de **`EDLinkedHashSet`**. ¡Ojo!, si usas *IntelliJ* Idea es posible que, al importar el fichero en tu proyecto, *IntelliJ* cambie automáticamente la expresión por

```
new Node[tamaño]
```

Esto es erróneo y el propio editor te lo indicará. Debes corregirlo usando la expresión inicial.
